

VeSPA Website Enhancements and VeSPAbot

Client:
Mark Nachazel
Virginia eSports Association

5/16/13
CS4624: Multimedia/Hypertext
Virginia Tech
Blacksburg

Jared Pepin
Gregory Jenks
Andrew Rothwell
Derek Richard

Table of Contents

Section 1: Executive Summary

Executive Summary_____	1.1	1
------------------------	-----	---

Section 2: User Manual

VeSPA Website

Introduction_____	2.1.1	2
Getting Started_____	2.1.2	2
Figure 2.1: The Users tab_____	2.1.2	2
Updating the Menu_____	2.1.3	3
Figure 2.2: The Header Menu_____	2.1.3	3
Using the Membership Plugin_____	2.1.4	3-5
Figure 2.3: The Membership Tab_____	2.1.4	4
Figure 2.4: Add New Buttons_____	2.1.4	4
Figure 2.5: Page Popups_____	2.1.4	5
Updating the Front Page_____	2.1.5	5-6
Figure 2.6: Widget Selection_____	2.1.5	6
Uploading Images_____	2.1.6	6
Sending out an Email Newsletter_____	2.1.7	6-7
Figure 2.7: Email Newsletter Tab_____	2.1.7	7
Updating the Login Page_____	2.1.8	7
Adding New Pages_____	2.1.9	7-8
Figure 2.8: Deactivating Mingle Forum_____	2.1.9	8
Figure 2.9: Add New Pages_____	2.1.9	8
Adding Plugins_____	2.1.10	8
Figure 2.10: Add New Plugins_____	2.1.10	8
Updating Plugins/Themes/WordPress_____	2.1.11	9

VeSPA Bot

Notice_____	2.2.1	9
Warning_____	2.2.2	10
Overview_____	2.2.3	10
Requirements_____	2.2.4	10
Usage_____	2.2.5	10-11
Modules_____	2.2.6	11-12
Table 2.1: Current Modules and Commands_____	2.2.1	11-12
Termination_____	2.2.7	13

Section 3: Developer Manual

VeSPA Website

Introduction_____	3.1.1	14
Inventory of Data Files_____	3.1.2	14-15
Table 3.1: Inventory of Files_____	3.1.2	14-15

Creating Custom Pages_____	3.1.3	15-16
Figure 3.1: Template Select_____	3.1.3	16
Adding Custom User Fields_____	3.1.4	16-18
Finding User by Twitch Name Field_____	3.1.5	18-19
Figure 3.2: The Query Window link_____	3.1.5	18
VeSPA Bot		
Overview_____	3.2.1	19
VeSPAbot Design Overview (developer view)_____	3.2.1.1	19
Figure 3.3: Class Diagram_____	3.2.1.1	20
Module Based Design_____	3.2.2	20
Adding Functionality_____	3.2.3	20-21
Class Documentation_____	3.2.4	21-24
Table 3.2: TwitchBot_____	3.2.4	21-22
Table 3.3: VespaBot_____	3.2.4	23-24
Developing VeSPAbot Modules		
Overview_____	3.3.1	24
Attributes_____	3.3.2	24-27
Commands_____	3.3.2.1	24
Setup_____	3.3.2.2	24
Privilege_____	3.3.2.3	25-26
Usage_____	3.3.2.4	26
Register_say_____	3.3.2.5	26
Register_terminate_____	3.3.2.6	26-27
Testing_____	3.3.3	27
Section 4: Our Process		
Timeline/Schedule_____	4.1	28-29
Problems_____	4.2	29-30
Solutions_____	4.3	30
Section 5: Acknowledgements and References		
Acknowledgements_____	5.1	31
References_____	5.2	31

Executive Summary

For our term project for our Multimedia/Hypertext capstone, our group took the task of revamping the Virginia eSports Association's website, www.vespa.org.vt.edu. After looking at their website extensively, we decided the focus of our updates would be on providing better publicity for the organization and making a much more usable interface with a few key features. VeSPA wanted us to keep the website based on a Wordpress blog to promote simple upkeep and updates. In addition, they wanted us to create a "Bot" to help with the chat monitoring and manipulation during streaming sessions.

To start off with, we made quite a few changes to the website to provide better security and appearance, and added new features to promote ease of use on the website in general. One of our main goals was to hide most of the WordPress identification on the site at VeSPA's request. To accomplish this, we removed standard WordPress features from the header of the webpage and changed the login page to display a VeSPA logo instead of a WordPress logo. This way, except for the admin dashboard, the website doesn't have direct links or features that visually identify it as a Wordpress site, (except for a small logo on the bottom of the page). Since VeSPA is not only a Virginia Tech school organization, using a club listserv would not be sufficient for disseminating emails to all VeSPA members, so we decided to add email newsletter functionality to the site. We made it so that by signing up for the website, you were added as a "subscribed user" for the email system and that your twitch name was added to the database so that, with added functionality, your twitch name could be used for certain raffle features.

Another big feature we added was the membership functionality which allows users to create accounts on the VeSPA website so they can view some pages that non-members cannot. The Forum was another feature we added which allows users to post messages and communicate with other VeSPA members about any topic they choose. One of our biggest accomplishments was creating a video gallery page that displayed all viewable Twitch, (a video streaming service), video files and Youtube video files belonging to VeSPA. A more general task that we accomplished was generating lots of additional content for the website and updating the information and images consistently for the webpage over the duration of the semester. This task required that we update information about tournaments, current events, set up sign up forms, and kept the site presentable.

Since streaming eSports games is a big part of what VeSPA does, providing a bot to manage the chat during streams was very important. To fulfill VeSPA's needs, we created the VeSPAbot, which is a specialized IRC bot that runs on VeSPA's Twitch channel and provides chat moderation and user interactions. The bot is modularized to allow for easy extension of its functionality and possible integration with the VeSPA website.

This document provides a user manual, which contains information for whoever will take over day to day administration of the website, a developer manual, which has some information about how to extend the functionality we've created, and a process section which describes our schedule and amount of work we did along with some problems we ran into and the solutions to those problems.

User Manual

VeSPA Website

Introduction

Since our project was to enhance the functionality of the Virginia E-Sports Association website, this “user manual” will not be a set of instructions for all users of the site, but will instead be a set of instructions for future site admins who might not have an extensive background in web development. It contains instructions for performing minor site duties and keeping the changes we’ve put in place running. For help in developing new, non-standard content and building on the functionality we’ve implemented, see the Developer Manual.

Getting Started

Due to the changes we made to remove some of the visual features included in WordPress pages, accessing the admin panel now requires a user who has admin privileges to log in the same way as any other site user.

To give a site member admin privileges, a different user must already have admin privileges. Once an admin user is logged in, they can access the admin panel by clicking the “admin” link at the top right of the page. The admin can then give other users admin privileges by going to the Users tab on the left side of the admin panel.

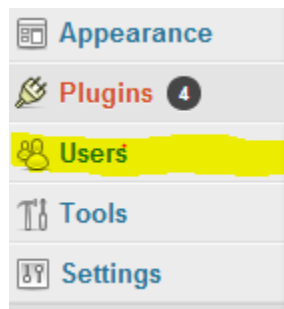


Figure 2.1: The Users tab

This will open up a page which lists all of the users registered to the site. Clicking on the user that should be given admin privileges allows the admin to edit their profile information. To make that user an admin as well, use the dropdown box labeled “Role” in the “Name” subcategory to change their role to Administrator. Then, scroll down to the bottom of the page and click Update User. That user now has admin privileges. Once all users who must be able to make changes to the site have been added as admin users, everyone can access the features described in this manual.

Updating the Menu

On every page of the VeSPA website, the header contains a horizontal list of the pages one can access on the website.

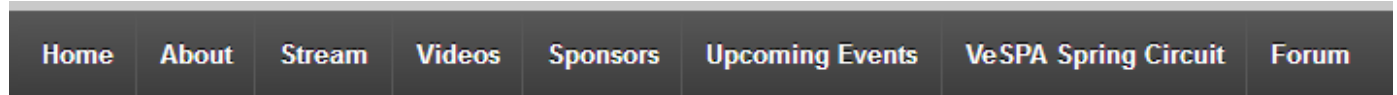


Figure 2.2: The Header Menu

To add a new link to this top menu, go to the Appearance tab on in the admin bar and then go to the Menu subpage. This should pull up a page which shows a listing of all the pages and subpages listed on the menu, which is called “VeSPA Menu”. To add new menu tabs, use one of the sections on the left side of the page to either select an existing page to add, add a custom link by inputting the page’s url, or click to select categories or media categories. The pages shown in the VeSPA Menu tab can be rearranged by dragging and dropping them in the desired order. Pages that are indented are nested within the page immediately above it, and will only be visible on the menu by hovering the mouse over the top-level page. Pages can be removed from the VeSPA Menu by expanding the page tabs by clicking the arrow on the right of the box and then clicking the Remove link on the bottom of the expanded page tab. When a new tab is added to the menu, you must change the options for the various membership levels to make the new tab visible (see the section titled “Using the Membership Plugin”).

Using the Membership Plugin

The membership plugin has a lot of configuration options that allow for a high degree of control over the membership aspects of the VeSPA site. In order to access these customization options, an admin user must be given Membership Administrator access in addition to their site administrator status. To make a user a Membership admin, any site administrator can go the Users tab on the left of the admin page and select the user to be given Membership Admin access. On their profile page, scroll down to the bottom of the page, where there is a section labeled “Membership Administration”. Checking the checkbox in that section and then clicking “Update User” gives that user full membership administration access.

When logged into the admin panel as a membership administrator, there should be a new tab at the very top of the admin bar on the left labeled “Membership”.

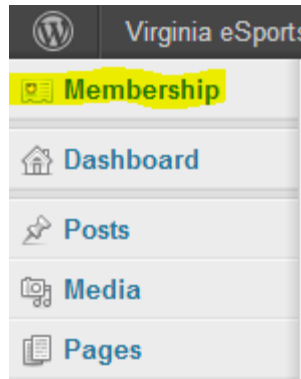


Figure 2.3: The Membership Tab

This tab opens up to reveal several sub-pages, which each allow customization of a different aspect of the membership plugin. One can view all of the members of the site by going to the “Edit Members” tab. From that page, it is possible to remove users from the site, edit their information, or change their membership level or subscription level.

The membership plugin allows admins to set certain pages to only be viewable by users who have a certain membership level. For example, if one wanted to create a new membership level for VeSPA Staff that could access pages that normal members could not, a few steps would be required.

First, to add the new member level, “VeSPA Staff”, go to the “Edit Levels” subpage inside the “Membership” tab. Then click the “Add New” button on the right.

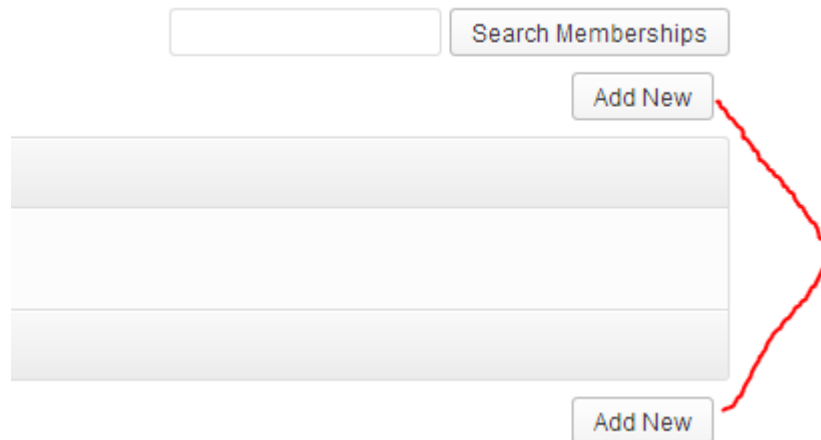


Figure 2.4: Add New Buttons

This will open up a page that allows you to name the new level and set basic configuration options. As these can be set later, leave them blank for now and set the name to “VeSPA Staff”. The free version of the Membership plugin only allows a maximum of two different membership levels. When users register for the site, the basic subscription will set their default level to the original “Site Member”, meaning all users on the VeSPA staff must be given the user level manually.

To manually change a user's membership level, go to the "Edit Members" page and hover over the section for the user to modify. Links should pop up below the Membership Level item labeled Add, Move and Drop. Click the Move link. On the page that comes up, select that user's current membership level in the first dropdown box and the "VeSPA Staff" level in the second box and then click the Move button on the bottom right.

To control the pages that certain user levels can access, go to the "Edit Levels" tab. Hovering over the "VeSPA Staff" level will make links pop up labeled Edit, Deactivate, Clone, and Delete.



Figure 2.5: Page Popups

Click the "Edit" link and use the menu and page tabs to select the pages and menu items that the membership level can access. Clicking the "Update" button at the bottom of the page will perform these changes. You can then use the same method to ensure that the other membership level cannot access the pages, by unchecking the pages and menu items that shouldn't be visible for that level.

Updating the Front Page

In order to update the front page, you must change a few different sections to fully update the entire page. The sections are as follows.

- For Updating the Header – Go to the appearance Tab on the left bar and under that menu, select Themes from the submenu.
 - Under "responsive child" at the top of the page, select customize underneath and this will take you to the customization menu.
 - Under the submenu, you can select Header image section and choose which image will be displayed at the top of the home page.
- For Updating the Widgets on the Front page - Go to the Appearance Tab on the left bar under the menu, select Widgets from the submenu.
 - On the right of the screen under the Home Widget 1, Home Widget 2, and Home Widget 3 submenu, you can choose from which of the "Available widgets which you would like to appear in those Home Widget slots, by dragging the selected "Available Widget" into the chosen "Home Widget" slot.



Figure 2.6: Widget Selection

- For Updating the Center of the Front Page - Go to the Appearance Tab on the left bar under the menu, select Theme Options from the submenu.
 - Click on the Home Page submenu to bring up all of your options.
 - You can change the Headline and SubHeadline from the selected menu which is the largest words you see on the front page.
 - Content Area can also be changed to modify the text that appears under the Headline and SubHeadline as well.
 - The “Call the Action URL” takes a given URL and creates a button that when clicked will take you to said website.
 - “Call the Action Text” changes the text that appears on said button.
 - Featured content is what is currently the image on the front page. It takes HTML code and does whatever the html requests. In our case for the website, we have chosen a particular image to display.

Uploading Images

To upload images to the website, go to the Media tab on the left sidebar. At the top of the page, to the right of the “Media Library” label, select the “Add New” button. Then on the next page, either drag an image or select an image to upload to the media library. There will usually be an “HTTP Error”, so just refresh the page and go back to the Media tab. The image will have uploaded even though the error appeared.

Sending out an Email Newsletter

To send an Email Newsletter out to users. Follow a few steps. First, Select Email newsletter from the bar on the left.

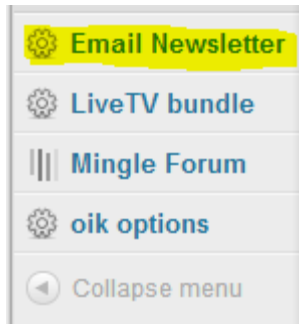


Figure 2.7: Email Newsletter Tab

Now select “Compose Email” from the submenu. Give the email a subject and for the email body, you must input HTML code. To make this easier we have provided a website to assist in this. The website provided is - <http://www.quackit.com/html/online-html-editor/>. Enter the text in the format you want in the main box to create the newsletter the way you want it to appear. Then at the top of the tools dashboard, select the “source” button. Then copy all the code that is now displayed and paste it back in the “email body” back on your WordPress page but erase the <html> and </html> tags. Then select Yes from the Display Status dropdown.

Now select insert Message. This will save the email to your WordPress account. To send out this email now, because of the way the registration to the website works, all users who sign up are added to the subscribed users list; select “send mail to subscribed users” from the submenu. From here you can choose exactly who this email goes out to, by default it is sent to everyone. Underneath, select the email you wish to send out and then select “send email” and the email will be sent.

To remove subscribed users, select “view subscribed users” from the submenu. Then check any users boxes to select the users you wish to remove. Then select “delete multiple records” to remove them from the subscribed users list.

Updating the Login Page

Go to the Settings tab on the left side. From this menu, select “Custom Login” from the submenu. From here, you can edit the login page however you would like. The background image is the image that appears above the login boxes. All other boxes change either the colors of the background of the page or the color based on hex values.

Adding New Pages

To add new pages to the website, you must first deactivate the Mingle-Forum. First, you need to select plugins from the bar on the left side of the page. Now go to the Mingle-Forum plugin and select deactivate.

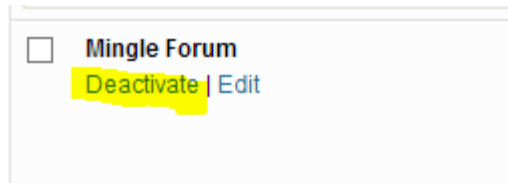


Figure 2.8: Deactivating Mingle Forum

Now go to the pages tab on the bar on the left of the screen. On the top of the screen, select the “add new” button.

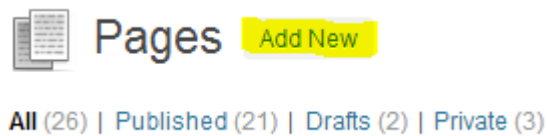


Figure 2.9: Add New Pages

You can now add whatever you like to the box. In the visual tab, what will appear on the web page is displayed. The textbox is where the HTML will appear and the HTML that you can edit. To take off the side widgets on the page, on the right side of the page, go to the Template drop down box and select “Full Width Page (no sidebar)”. Under the Discussion bar farther down the page, uncheck the “Allow Comments” box to not allow comments on the page when it is on the website. Then on the right side of the screen inside the publish box, hit the “publish” button to make the page viewable. Make sure the new page is viewable to the selected member level, so view the membership section to see how to set your new page to be viewed by who you would like. Re-activate the Mingle-Forum plugin the same way you deactivated it.

Adding Plugins

To add plugins to the site, select the plugin tab on the bar on the left side of the screen. Now select the “add new” button.



Figure 2.10: Add New Plugins

From here, you can add plugins one of two ways. The easiest way is to go use the search bar to search for the plugin that you wish to add. Now with the app you wish to add, click install now. This should install the plugin for your use. If this doesn’t work, you can go to the download the zip file of the plugin you want to download from the associated plugin web page. Then from the same add new tab, and at the top of the page, select the upload link which will allow you to manually install the plugin from the selected zip file. Now to activate whatever plugins you have installed go back to the plugins main menu and identify the plugin you have just installed. Now select activate and the plugin will now be active on the web page.

Updating Plugins/Themes/WordPress

Note: These Plugins must stay active:

BM Custom Login
Category Posts Widget
Contact Form 7
Disable Admin Bar
Email newsletter
LiveTV Team - 0 - Main Fork
liveTV Team - 1 - General shortcode
liveTV Team - 2 - Display lives
liveTV Team - 3 - IRC chat
liveTV Team - 4 - Widget
Media Category Library
Membership
oik-nivo-slider
oik base plugin

As we have made changes to the code for certain plugins and to the Responsive Theme, if these things are updated, it will be necessary to update certain plugins. When updating plugins, most should work fine, but there are certain plugins that we specifically modified that need to have their code re-modified after the update. These files should be packaged with this document. You must contact Andy Hsu or the VT Web Hosting site to gain access to the current file system to re-implement these changes.

wp-login.php:

```
$login_header_url = __( 'http://www.vespa.org.vt.edu' );  
$login_header_title = __( 'Virginia eSports Association' );
```

The left side of the “=” should still be there, but the right side needs to be changed

For updating the other required plugins, see the table located at the beginning of the “Developers Manual”. This should explain what code changes need to be made in order to modify the plugins after updates.

VeSPAbot

Notice

The instructions within this document are meant for Windows platforms (specifically Windows 7). However, these instructions should not differ radically between Windows, UNIX, or Mac based systems.

Warning

The current state of the VeSPAbot has only been tested on Windows 7.

Overview

The VeSPAbot is meant to be a specialized IRC bot that works within the VeSPA chat on the VeSPALive Twitch page located at <http://www.twitch.tv/vespalive>. The bot provides some basic functionality, such as a producer-consumer message distribution pattern aimed at dodging the built-in Twitch chat spam filter, a list of recognized Twitch accounts of members of and streamers for the VeSPA organization, and the capability to respond to commands posted into the chat by users.

All the files currently included are: VespaBot.py, twitchBot.py, parseModulesFromFile.py, VespaBotExample.cfg or VespaBot.cfg, builtinregulars.py, builtinowners.py, join_queue.py, message_threads.py, raffle.py, and RaidCall.py. The join_queue.py, message_threads.py, raffle.py, and RaidCall.py files are all module files, which are files used to add specific capabilities to the bot when it is run. Any number of the modules can be loaded in any order, and the process for doing so is described in the 'Usage' section. Additionally, it is possible to develop more modules as described in the document "How to Create a Module for VeSPABot."

Requirements

Running an instance of VeSPAbot requires Python version 3.X (3.3 being the latest version of Python during development of this version of VeSPAbot). Details and instructions on how to install Python can be found at <http://www.python.org>.

If you obtained a copy of VespaBotExample.cfg then you will have to create your own account on Twitch.tv to use in the settings section and will have to remove or replace the redacted names in the streamers section. Note that Twitch frowns upon having a bot join a channel without permission from the chat owner.

Usage

To run VeSPAbot on your system, open the Windows command window and change the directory to the folder containing the VeSPAbot files (all the files, listed in the 'Overview' section, should be in the same folder). You can also open the folder containing the files, shift+right-click, and select "open command window here." In the command window, enter one of the lines discussed below and hit enter.

Simply inputting "python VespaBot.py" will print out the usage message for the bot. The bot is designed to be run with a list of modules that extend its functionality. As it stands on its own, the main 'body' of the bot is just a shell that handles IRC and network connection protocols and respond to commands defined in modules. To run the bot with a specific list of modules, input either "python VespaBot.py -l module1.py module2.py . . . module.py" or "python VespaBot.py --list module1.py module2.py . . . module.py." Additionally, you can create a text file which contains the name of the modules you wish to use, one per line, and use that file instead of manually listing the files each time you run the bot. The input for this method is

“python VespaBot.py -f filename.txt.” It is also acceptable to use “--file” in place of “-f.” It is strongly recommended to run the bot using at least the builtinowners.py and builtinregulars.py modules for basic functionality.

While running, the bot will print all the messages it receives to the command window.

Modules

The current version of VeSPAbot includes the modules join_queue.py, message_threads.py, builtinregulars.py, builtinowners.py, raffle.py, excitedBot.py, and RaidCall.py.

Commands	Description
join_queue	Allows users to a join a queue most likely to get into the next game. Owners can get people out of the queue, or kick them from the queue.
!joinQueue <i>name</i> –any users; join the queue !getPlayers 1-9 –owners only; get people from queue !removePerson <i>name</i> –owners only; kick someone from the queue !exitQueue –any user; remove yourself from the queue	
message_threads	Allows owners to create messages that automatically re-post to the chat after a given time.
!spawnThread <i>interval message</i> –owners only; spawn a thread to display the message every <i>interval</i> seconds	
Builtinregulars	Basic functions for regular users: getting the current streamer, a list of commands, and a help function.
!streamer –any user; get a blurb about the current streamers !commands –any user; get a list of commands available to regular users !help <i>command</i> –any user; get a help	

message for the command if available	
Builtinowners	Basic functions for owners: setting the streamer, changing the shush mode, and shutting down the bot.
!toggleShush –owners only, toggle the bots ability to speak in chat !shushStatus –owners only, get the current shush mode (on or off) !setStreamer <i>name</i> –owners only, set the current streamer, must be a streamer listed in the cfg file !quit –owners only; safely terminate the VeSPAbot program	
Raffle	Allows users to enter a raffle, owners to randomly select a winner, and saves winners to a text file for future reference.
!joinRaffle –any user; enter the raffle !pickWinner –owners only; get a winner from the raffle and write the winners Twitch name and the date and time they won to a text file, winners.txt !quitRaffle –any user; remove yourself from the queue !raffleKick <i>name</i> –owners only, kick a person from the raffle	
excitedBot	Uppercases every outgoing message from the bot. Enabled by default.
!enableUpper –owners only; turn on all caps !disableUpper –owners only; turn off all caps	
RaidCall	Gives the Vespa RaidCall server number.
!raidcall –any user; get the VeSPA Raidcall server number	

Table 2.1: Current Modules and Commands

Termination

The bot can be closed by simply closing the command window it is running in. However, it is recommended to have an owner account registered with the bot in the VespaBot.cfg file use the command “!quit” in the Twitch chat in order to safely close all the threads the bot is using in the background.

Developer Manual

VeSPA Website

Introduction

The majority of the features we implemented for the VeSPA website were not meant to be extended and were simply designed to be kept up using the admin information in the User Manual. This manual describes a few key places in the site which could be expanded.

In general, whenever new functionality is needed for the VeSPA website, the first step would be to check if a WordPress plugin exists which provides the desired functionality. If one does not exist and the new functionality could be implemented on a single page, such as the Twitch and Youtube Video page we implemented, refer to the section titled “Creating Custom Pages”. If extensive changes must be made to a plugin or WordPress in general, there is a vast wealth of knowledge about WordPress development on the internet.

Inventory of Data Files

File Name	Location of the file on vespa.org.vt.edu	Explanation of Code
twitch_youtube_player.php	wp-content/themes/responsive/	Self-contained page which uses both the Twitch and YouTube API to get lists of videos and has an embedded player for both YouTube and Twitch videos. Whenever the Responsive theme is updated this file will have to be placed back into the given directory. Also, for this page to work properly, do not remove the file in the home directory labeled jquery-1.9.1.min.js.
account.form.php	/wp-content/plugins/membership/membershipincludes/includes/account.form.php	Contains two sections of code which contribute to the changes made to allow for the custom Twitch username field and email newsletter subscription control. When the Membership plugin is updated, the two code sections must be re-inserted (at their current positions in the file) into the file at the given location.
functions.php	/wp-content/themes/responsive/includes/functions.php	Contains changes that contribute to the Twitch username custom field, removal of the admin bar, the email newsletter functionality, and the

		custom login bar at the top of every page. Whenever the Responsive theme is updated the code in this file will have to be re-inserted anywhere in the functions.php file located at the location given on the left.
registration.form.php	/wp-content/plugins/membership/membershipincludes/includes/registration.form.php	Adds the option to register for email newsletters to the account registration page. Whenever the Membership plugin is updated the code section in this file must be reinserted into the file at the given location.

Table 3.1: Inventory of Files

Creating Custom Pages

To create a custom page that uses PHP and/or JavaScript can be difficult within the bounds of the page creation tool in the WordPress admin section. It is simpler to create one using a similar method to what was used for the twitch/youtube player, (included in the file twitch_youtube_player.php). Instead of editing a page in the WordPress editor, simply create any normal HTML/PHP page you want and add the following lines at the top and bottom.

```

<?php
/*
Template Name:      !! Name Your Page Here !!
*/
?>

<?php get_header(); ?>

    !! Your custom page goes here !!

<?php get_footer(); ?>

```

Then, place the file you created with the above header and footer into the directory.

wp-content/themes/responsive/

Then, create a new page in the WordPress. Give it whatever title you would like at the top of the page and on the right side of the text-editor expand the Template dropdown box and select the list item that matches the name you put in at the top of your page.

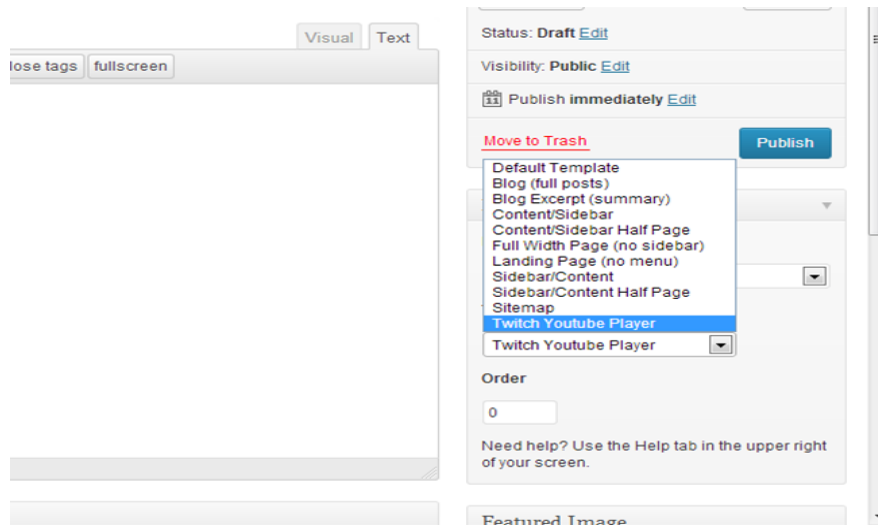


Figure 3.1: Template Select

Do not put anything in the page's text editor section. When this page is saved, the content in your original file will be displayed inside the standard WordPress header and footer. Make sure to add the page to the menu at the top of the page, as described in the "User Manual" section labeled Updating the Menu.

Adding Custom User Fields

When we were adding the membership functionality to the site, it occurred to us that the user profile fields that existed didn't tell us much about the users in an E-Sports context. We thought that it would be interesting to give site members the option to add other user fields, such as their Steam username, League of Legends account name, or Twitch username so that Vespa could utilize that information. We implemented the addition of the Twitch username. The file listed above named account.form.php contains a new chunk of HTML code that allows users to see and edit the Twitch username field. The rest of the code that handles the Twitch username is in the functions.php. A section in functions.php is labeled with comments above and below it; that php code handles the backend.

To add a new custom user field, one would simply duplicate the section of the code particular to the twitch name in the functions.php file, changing every instance of "Twitch Name" to the value of the new field. For example, the section of the code in functions.php would look like...

```
function add_twitch_name_field( $user ) {
?>
<h3><?php _e('Extra Profile Information', 'membership'); ?></h3>

<table class="form-table">
<tr>
<th>
<label for="address"><?php _e('Twitch Name', 'membership'); ?>
```

```

        </label></th>
        <td>
            <input type="text" name="address" id="address" value="<?php echo esc_attr(
get_the_author_meta( 'twitch_name', $user->ID ) ); ?>" class="regular-text" /><br />
            <span class="description"><?php _e('Twitch Name', 'membership'); ?></span>
        </td>
    </tr>
</table>

<table class="form-table">
    <tr>
        <th>
            <label for="address"><?php _e('New Custom Field', 'membership'); ?>
        </label></th>
        <td>
            <input type="text" name="address" id="address" value="<?php echo esc_attr(
get_the_author_meta('new_field', $user->ID ) ); ?>" class="regular-text" /><br />
            <span class="description"><?php _e('New Custom Field', 'membership');
?></span>
        </td>
    </tr>
</table>
<?php }

```

```

function save_twitch_name_field( $user_id ) {

    if ( !current_user_can( 'edit_user', $user_id ) )
        return FALSE;

    update_usermeta( $user_id, 'twitch_name', $_POST['twitch_name'] );
}

```

```

function save_new_field( $user_id ) {

    if ( !current_user_can( 'edit_user', $user_id ) )
        return FALSE;

    update_usermeta( $user_id, 'new_field', $_POST['new_field'] );
}

```

```

add_action( 'show_user_profile', 'add_twitch_name_field' );
add_action( 'edit_user_profile', 'add_twitch_name_field' );

```

```

add_action( 'personal_options_update', 'save_twitch_name_field' );

```

```
add_action( 'edit_user_profile_update', 'save_twitch_name_field' );
```

```
add_action( 'personal_options_update', 'save_new_field' );
```

```
add_action( 'edit_user_profile_update', 'save_new_field' );
```

(The bolded sections are the code that should be added, with “new_field” being whatever the desired name of the new user field)

A similar duplication would be necessary in the account.form.php file (in the HTML that adds the Twitch name to the account page), and the code would then have to be reinserted into the locations listed in the table above.

Finding User by Twitch Name Field

To find a Vespa site member based on their Twitch username, you would have to log on to the Vespa file system located at <https://secure.org.vt.edu/orgadmin/> (with the Web Server Address, www.vespa.org.vt.edu and your regular VT login and password, as long as you’ve been given permission). Once you’re logged in, click on the databases tab at the top of the page. On the database page, click on the Query Window link at the top of the scroll bar on the left of the page.



Figure 3.2: The Query Window link

In the query window, input the following query, with “TWITCH NAME” replaced with the desired twitch username to search for.

```

SELECT *
FROM `wp_users`
WHERE ID = (
    SELECT user_id
    FROM `wp_usermeta`
    WHERE `meta_key` = 'twitch_name'
    AND `meta_value` = 'TWITCH NAME' LIMIT 0,1)
LIMIT 0,1

```

The query will return the one row in the wp_users table that has the matching twitch_name metadata entry. This could be used in combination with VeSPABot's raffle.py functionality to create some sort of raffle which selects one user who was present during a stream and is also a member of the Vespa website.

VeSPABot, an IRC bot for the VeSPA Twitch channel chat

Overview

The VeSPABot is meant to be a specialized IRC bot that works within the VeSPA chat on the VeSPALive Twitch page located at <http://www.twitch.tv/vespalive>. The bot provides some basic functionality, such as a producer-consumer message distribution pattern aimed at dodging the built-in Twitch chat spam filter, a list of recognized Twitch accounts of members of and streamers for the VeSPA organization, and the capability to respond to commands posted into the chat by users

VeSPABot Design Overview (developer view):

VeSPABot's primary existence (its ability to join and listen to a chat) is located in the twitchBot.py TwitchBot class and the VespaBot.py VespaBot class. The TwitchBot class handles the IRC protocols and the actual connection to the vespalive twitch channel chat by subclassing the asynchat.asynchat class from the Python library. The VespaBot class subclasses TwitchBot and handles loading modules and responding to chat commands.

Modules are files that provided the actual functions the bot will use to respond to commands in the chat. More about how to create a module for VeSPABot can be found in the "How to Create a Module For VespaBot" document.

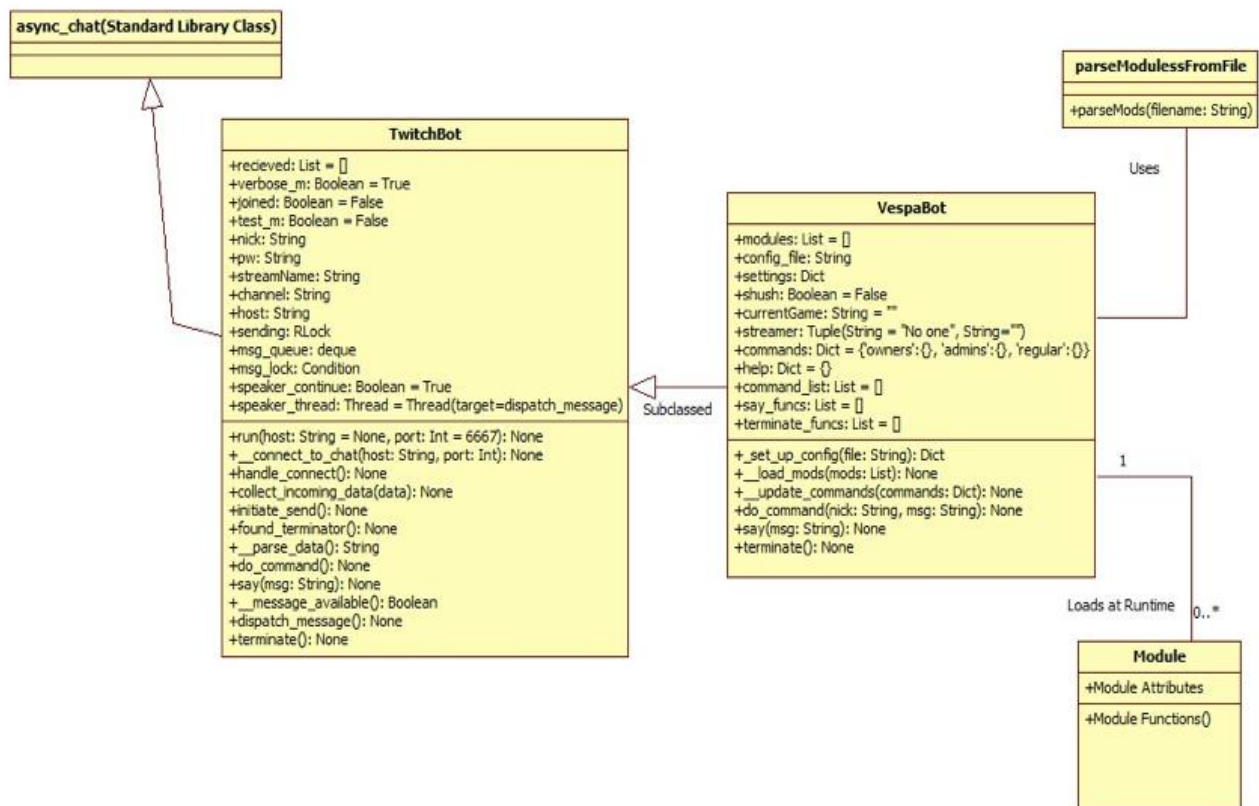


Figure 3.3: Class Diagram

Module Based Design

VeSPAbot is designed to have no innate functionality itself beyond parsing user input and running commands implemented in module files. VeSPAbot should be thought of as a container for data that might be made use of in current or future modules. For example, information about the identity of the current streamer is used by the builtinregulars.py module file, and can be reasonably expected to be used by future modules.

Adding Functionality

Chat functionality (the ability to perform functions based on user input into the chat room) should be implemented in module files to be loaded by an instance of VeSPAbot at runtime. However, fields can be added to the bot to support the module's functionality. If necessary, it is certainly possible to add additional functions to the bot to change its behavior. For example, the current version of VeSPAbot only superficially supports the ability to recognize chat admins and commands that can only be run by admins. Fleshing out this behavior fully would require modifying how the TwitchBot chat parses chat nicknames in the found_terminator

method and altering the `do_command` method in `VespaBot` to check commands of the “admin” privilege level. A full list of fields and methods available in the bots is provided in this document.

Class Documentation

<code>TwitchBot(nickname, passw, stream, verbose=True, test=False)</code>
Defines an instance of <code>TwitchBot</code>, a class designed to handle connecting to a Twitch.tv channel chat, parse the chat activity, and handle the IRC protocols. <code>TwitchBot</code> also implements a producer-consumer thread module designed to limit the bots messaging activity to at most one message every 3.1 seconds to avoid losing messages to the Twitch.tv chat spam filter. This bot is mainly meant to be subclassed and just serves as a protocol handling base class. <code>TwitchBot</code> subclasses the <code>asyncchat.asyn_chat</code> class. Running an instance of <code>TwitchBot</code> requires a twitch account name, a password for that account, and a channel name. Keyword arguments: <code>verbose</code>: a boolean value controlling print statements for debugging and possible logging purposes. Defaults <code>True</code>. <code>test</code>: a boolean value that controls whether output should made to <code>sys.stdout</code> or to the twitch channel chat. Defaults <code>False</code>.
<code>run(host=None, port=6667)</code>
Handles connecting to the Twitch.tv chat server. Keyword arguments: <code>host</code>=host name of the server. Defaults to an internally calculated name based on the stream name given on creation. <code>port</code>=port to connect to. Defaults to the current port specified by Twitch.tv, 6667.
<code>__connect_to_chat(host, port)</code>
Create a socket and connect to the chat.
<code>handle_connect()</code>
Sends the proper IRC formatted messages required to join the Twitch.tv channel chat.

collect_incoming_data(data)
Reads in incoming chat data.
initiate_send()
Attempts to makes the async_chat method of the same name thread safe.
found_terminator()
Recognize a recieved message and parse it to get the name and message and check to see if it is a command. Also responds to periodic 'ping' messages from the Twitch.tv chat server.
__parse_data()
parses an IRC message that has been recieved from raw chat data.
do_command(nick, msg)
Should be implemented by any subclass
say(msg)
Produces a message that, when consumed, will be pushed to the chat by adding the message to a queue.
__message_available()
Returns True is there is a message in the queue. Used as the signaling function for the dispatch thread conditions variable.
dispatch_message()
This function is run exclusively by an outside thread. Consumes an available message in the message queue and then sleeps before 3.1 seconds.
terminate()
Attempt to safely terminate the bot by waiting for the consumer thread to close.

Table 3.2: TwitchBot

VespaBot(configure_file, mods, verbosemode=True, testmode=False)

Defines an instance of VeSPAbot, an IRC bot designed for the VeSPA Twitch.tv channel chat that subclasses twitchBot.TwitchBot. The class requires a cfg file with the template and information described below and a list of module files to load at runtime:

```

[settings]
nick=bot's twitch account name
password=bot twitch account password
stream=the twitch channel name
owner= a list of twitch accounts the bot will recognize as 'owners' of the
bot

```

```

[streamers]
none=message to display if no streamer is set
streamer name 1: information to display for this streamer
...
streamer name N: information to display for this streamer

```

Keyword arguments:

- verbosemode: a boolean value controlling print statements for debugging and possible logging purposes. Defaults True.
- testmode: a boolean value that controls whether output should be made to sys.stdout or to the twitch channel chat. Defaults False.

__set_up_config(file)

Given a config file, file, parse it to get the nickname, password, stream name, and owners for the bot to use to login to the bot's twitch account

__update_commands(commands)

Parses the functions of a module and registers them to the VespaBot instance according to any special attributes the function has

do_command(nick, msg)

Given a user nickname and message, check to see if the message is an acceptable command and, if so, execute that command

say(msg)

Gateway function to TwitchBot.say function, applies and special functions registered through __do_commands and imposes the shush mode behavior to prohibit or allow bot messages

terminate()

Attempt to safely terminates the VespaBot instance by running any special functions registered through `__do_commands` and calling the `TwitchBot.terminate` function

Table 3.3: VespaBot

Developing VeSPAbot Modules

Overview

A VespaBot module is essentially a list of function definitions and any variables those functions rely on to operate. Any function that interacts directly with the running instance of VespaBot needs to have certain attributes in order for the VespaBot instance to know what to do with the function.

For example, take the simple module `RaidCall.py`:

```
RaidCall_number = "5580565"
```

```
def raidcall(bot, nick, msg):
    bot.say("Join our raidcall server: "+RaidCall_number)
    raidcall.commands=["^!raidcall$"]
```

The attribute “commands” contains a list of all regex patterns that the VespaBot instance should match chat member messages against to see if it should execute the raidcall function.

Attributes

Any function that interacts directly with the VespaBot instance, such as changing variables in the bot’s settings or using one of the bot’s functions (most often the ‘say’ function) will require a specific, or possibly multiple, attribute(s). The supported attributes are explained below.

Commands

This attribute is required for any function that is to be invoked based on the messages of users in chat.

```
def function(bot, nick, msg):
    #code
    function.commands=[list of matching regex]
```

Any function that attaches the command attribute must take in three parameters: a reference to the bot that called the function (‘bot’), the nickname in the chat of the user

who invoked the command ('nick'), and the message the user sent (msg). Whether these parameters are used is up to the programmer.

The list of matching regex should be a list with at least one regex pattern that the bot will use to know when a user wants it to invoke the function.

Setup

The "setup" attribute should be given to a function that should be called as the bot is starting up. The primary purpose of this attribute is to perform any initialization actions required for the module to operate properly.

```
def function(bot, nick, msg):  
    #code  
    function.setup=0
```

The value of the setup function does not matter, but for consistency it is advised to set the attribute to 0. It is recommended that the "say" attribute be the only one set on a function; however it is feasible for the function to also use the "command" attribute depending on the actions taken within the function.

****A function should use both the command and setup attributes****

Privilege

This attribute determines what type of chat member can invoke the command

```
def function(bot, nick, msg):  
    #code  
    function.privilege="owners", "admin", or "regular"
```

The privilege attribute should be set to one of three things: "owners", "admin", or "regular". Any functions with a "commands" attribute but not a privilege attribute is assumed to have the "regular" privilege.

Privilege refers to who in the chat has access to the function (or more specifically, access to the command used to invoke the function). The "owners" level means only members of the chat whose chat names are listed in the VespaBot.cfg file can invoke the command. This is typically reserved for functions that set important information or moderate other chat features. The "admin" level is only superficially supported at the moment, but would allow only chat users marked as chat administrators and members designated as owners to use the function. Finally, the "regular" level means any chat user can invoke the function.

Usage

The usage attribute is used to define a help message to be used with the "!help" command in the builtinregulars module. This attribute is meant to be used with a "command" function that makes use of the command attribute.

```
def function(bot, nick, msg):
    #code
    function.usage=("command_name", "usage_message")
```

The usage attribute should be set to a tuple; the first value in the tuple should be the name of the command (for user ease, this should probably match the function name) and the second value should be the usage message. The usage message should be a quick blurb describing how to use the command.

Register_say

The register_say attribute is used to tell the bot that this function should be called before the bot completes the “say” function which pushed messages out to the chat. The most direct way to use this attribute is to create a function that modifies the message about to be pushed to the chat, such as upper-casing the message.

```
def function(bot, nick, msg):
    #code
    function.register_say=0
```

The value of the attribute is irrelevant, but it needs to be given a value. As a standard, simply set the attribute to 0. The register_say function should be the only attribute given to the function

It is recommended that any module that includes a function or functions with the register_say attribute should include associated command functions that can enable and disable the register_say functions. In order to do this, you must access the list of functions the bot keeps to apply before it finishes the “say” function. The list can be acquired through the line “bot.say_funcs.”

Register_terminate

The register_terminate attribute signifies that a function should be called when the bot calls its “terminate” function. The purpose of this function is to provide a safe way to close any files, threads, or similar resources the module has opened.

```
def function(bot, nick, msg):
    #code
    function.register_terminate=0
```

The register_terminate function should be the only attribute given to the function. Like the “register_say” attribute, the value of the register_terminate does not matter. For consistency it is best to set the value to 0.

Testing

To test module functionality, you can create a test function in the `vbot_test.py` file and add the function to the call list in the “main” section of the file. Every test function should be preceded by the `@test` decorator. This wraps the test function with a test harness that handles printing out the results of the test. The test function should follow the general form below:

```
def testFunction(bot):
```

```
    #code
```

The test harness expects the function to return true or false based on whether or not the test passed. The general way to carry out a test is to use the `forceMessage` function in `test` file to send a message to an instance of `VespaBot`, wait a few seconds (this makes sure the message has time to trickle through the bot and get passed the spam filter dodger), then capture the output of the `VespaBot` instance using “`sys.stdout.getvalue()`.” An example test case for the `RaidCall.py` module would look like

```
@test
```

```
def testRaidCall(bot):
```

```
    forceMessage(bot, "pseudoBot", "!raidcall")
```

```
    time.sleep(3)
```

```
    return "Join our" in sys.stdout.getvalue()
```

Our Process

Timeline/Schedule

The total, (estimated), count of man-hours we put into the project are spread out as follows...

- Content Generation(Updating of Information, Videos and Posts) - 15 hr
- Custom Login Changes- 10 hr
- Youtube Plugin - 4 hr
- Email Newsletter - 15 hr
- Forum - 10 hr
- Image Creating and Editing - 2 hr
- Issue Solutions and Problem Solving - 10 hr
- Membership Plugin Install and Setup - 10 hr
- Membership Plugin Customization - 15 hr
- Toolbar Removal - 5 hr
- WordPress Backup - 2 hr
- In-House Game Contact Form - 2 hr
- Video Page - 58 hr
- To-Do List - 3 hr
- VeSPA Bot Version 1
 - Research - 8 hr
 - Development - 5 hr
 - Testing - 2 hr
- VeSPA Bot Version 2
 - Research 15 hr
 - Development - 10 hr
 - Testing - 5 hr
- VeSPA Bot Version 3
 - Research - 10 hr
 - Development - 15 hr
 - Testing - 10 hr
- Documentation - 20 hr

Total Time Spent – 261 hours

The timeline of our project was broken up into 3 sections.

Phase 1: Feb 4 - Mar 12

- VeSPA Bot Version 1
- Content Generation
- Image Creating and Editing
- In-House Game Contact Form
- Membership Plugin Install and Setup
- Membership Plugin Customization
- Gaining and understanding of Wordpress
- WordPress Backup

Phase 2: Mar 13 - Apr 12

- VeSPA Bot Version 2
- Content Generation
- Toolbar Removal
- Forum
- Email Newsletter
- Custom Login Changes

Phase 3: Apr 13 - May 8

- VeSPA Bot Version 3
- Content Generation
- Youtube Plugin
- Video Page
- To-Do List

Problems

We encountered a few problems over the developmental phase of this project. Some were simple issues that we found workarounds for while others were more complicated and limited our results.

- The first issue we encountered was that no one in our group had any knowledge of PHP before we started, and almost all plugin changes we needed to implement were PHP based changes.
- We had to document and figure out how to keep our changes to the plug-ins after site updates.
- The website was based on a WordPress blog, and we were limited in how we could change the site and how we could adapt it for our needs. This also meant that we had to learn WordPress site management before we could even start any coding.
- We ran into a few odd issues with file editing, because there is no change log supported on the VT hosting site, so if anything went wrong or code was added incorrectly into some file, there was no way of backtracking what we had done. We occasionally had problems where code from one file would be inexplicably placed into other files during editing, (apparently due to some sort of glitch involving the back button), causing massive issues that were very hard to fix.
- Probably the biggest issue we found was that there is a very small PHP limit provided to us org.hosting.vt.edu. This meant that we weren't able to use certain plug-ins in conjunction with others. And because the forum plug-in we used takes up so much memory, we had to be very economical in the way we selected which plugins we could and could not include.
- VeSPAbot is based on Python3.3, which no one was very familiar with so a lot of time was spent checking basic syntax and library functions
- Since no one was that versed in Python, it wasn't until later we discovered the `async_chat` module that made connecting to the Twitch chat service much easier.
- However, that same module left out documentation of some important functions which lead to time consuming debugging (for example, the documentation does not mention the very important `initiate_send` method).
- Combining threading with the `async_chat` module proved to be difficult, especially when it came to properly closing the bot and trying to catch errors. This problem was magnified by the `message_threads` module which allows chat owners to spawn unlimited

amounts of new threads.

Solutions

We were able to fix most of the problems we ran into.

- We learned PHP for modifying the website, through use of Google and searching for examples that we could use to help show us how to edit the plug-ins as needed.
- To keep our changes to the plug-ins after updates, we put all the code that we added in separate files and then documented where you would need to add the code and into what files to keep our changes working in the “Developer’s Manual” section.
- We spent some time and used trial and error to figure out how to modify the site and front page. We did the same to figure out how to add and modify new pages and posts.
- There was no real solution to the file editing issue. All we did was keep searching through every file we had touched and looked for anywhere we could have added code to take out to fix the issues we were having.
- Most of the solutions for the problems with VeSPAbot were solved with help from stackoverflow.com, reading and rereading python documentation pages, and by referencing an example Python IRC bot available on GitHub.

Acknowledgements and References

Acknowledgements

Mark Nachazel

Email: marknach@vt.edu

Phone:(703) 244-3061

We'd like to thank Mark for giving us the opportunity to work on the site and helping to get us started.

References

Dickson, Ian. (2000). Quackit. Retrieved from www.quackit.com

Mutantmonkey (2013) *Phenny* [Computer program]. Available at <https://github.com/mutantmonkey/phenny>

Python Software Foundation. (1990). Python Programming Language – Official Website. Retrieved from www.python.org

Phenny is an open source, general IRC bot designed to join multiple channels and run various different modules. It was an excellent reference for learning how to connect to an IRC server as well as how to implement a dynamic module structure.